Automated Software Fault Measurement

Allen P. Nikora
Jet Propulsion Laboratory,
California Institute of Technology
Pasadena, CA
Allen.P.Nikora@jpl.nasa.gov

John C. Munson
Computer Science Department
University of Idaho
Moscow, ID
Jmunson@cs.uidaho.edu

The work described in this paper was carried out at the Jet Propulsion Laboratory, California Institute of Technology. This work is sponsored by the National Aeronautics and Space Administration's Office of Safety and Mission Assurance under the NASA Software Program led by the NASA Software IV&V Facility. This activity is managed locally at JPL through the Assurance Technology Program Office (ATPO).

Agenda

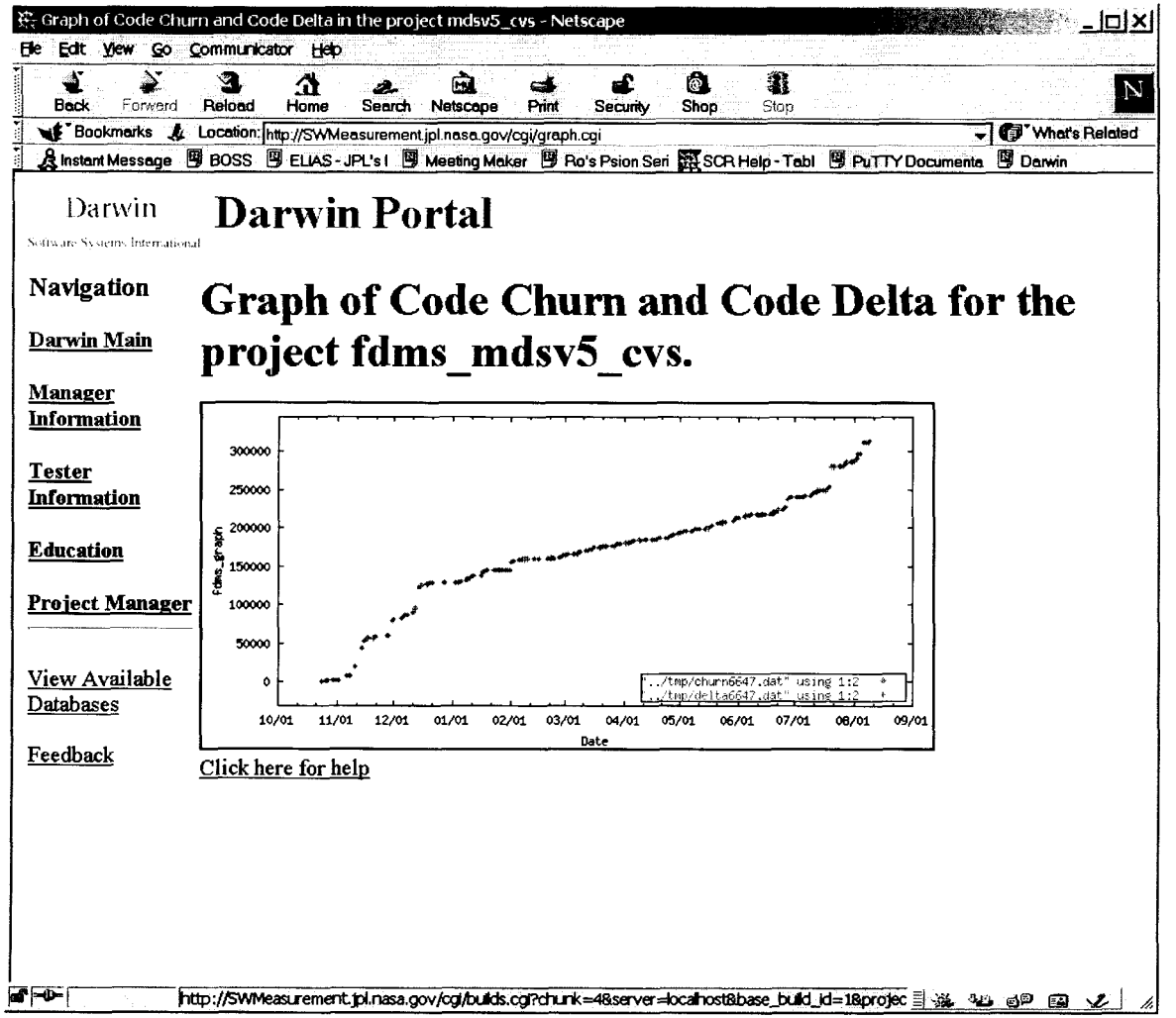
- Motivation
- Current State of Affairs
- Approach
  - Examples
- Identifying and Counting Faults
  - Examples
- Current Work

Motivation

- One goal of current work is to improve understanding of the type of faults that are inserted into a software system during its lifetime by identifying relationships between types of structural change and the number and types of faults inserted.
- Developing software fault models depends on definition of what constitutes a fault
- Desired characteristics of measurements, measurement process
  - Repeatable, accurate count of faults
  - Measure at same level at which structural measurements are taken
    - Measure at module level (e.g., function, method)
  - Easily automated

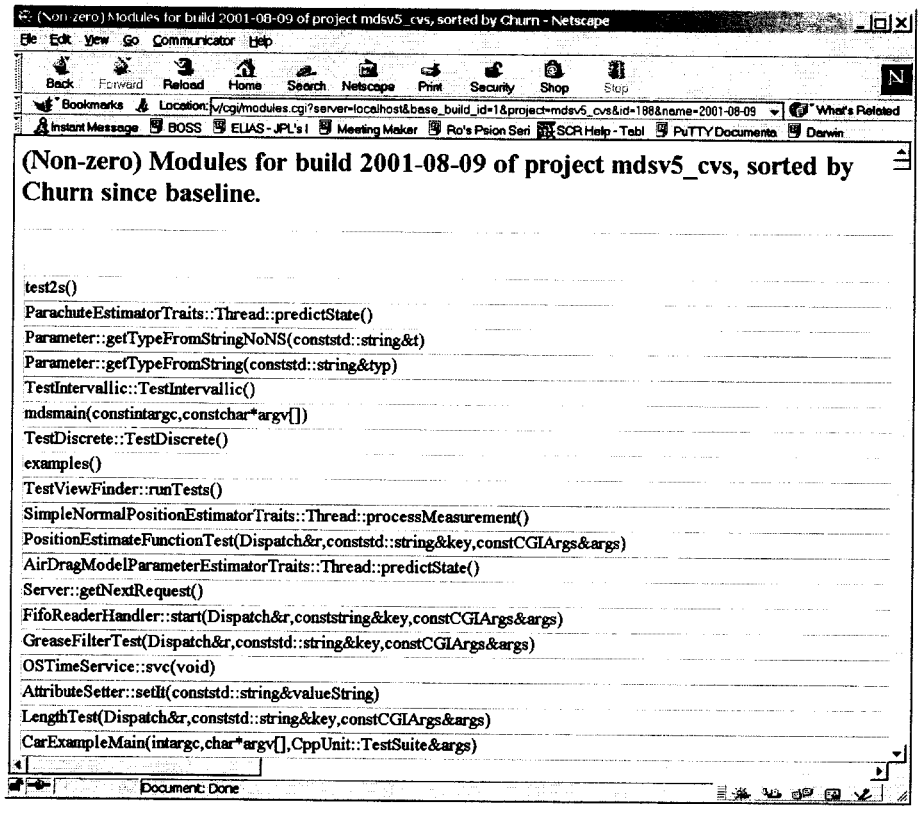
Motivation (cont'd)

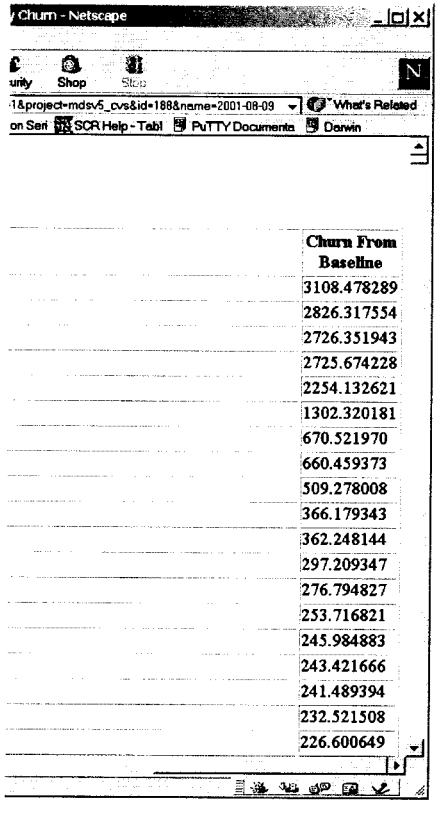
Measurement of Structural Evolution



Motivation (cont'd)

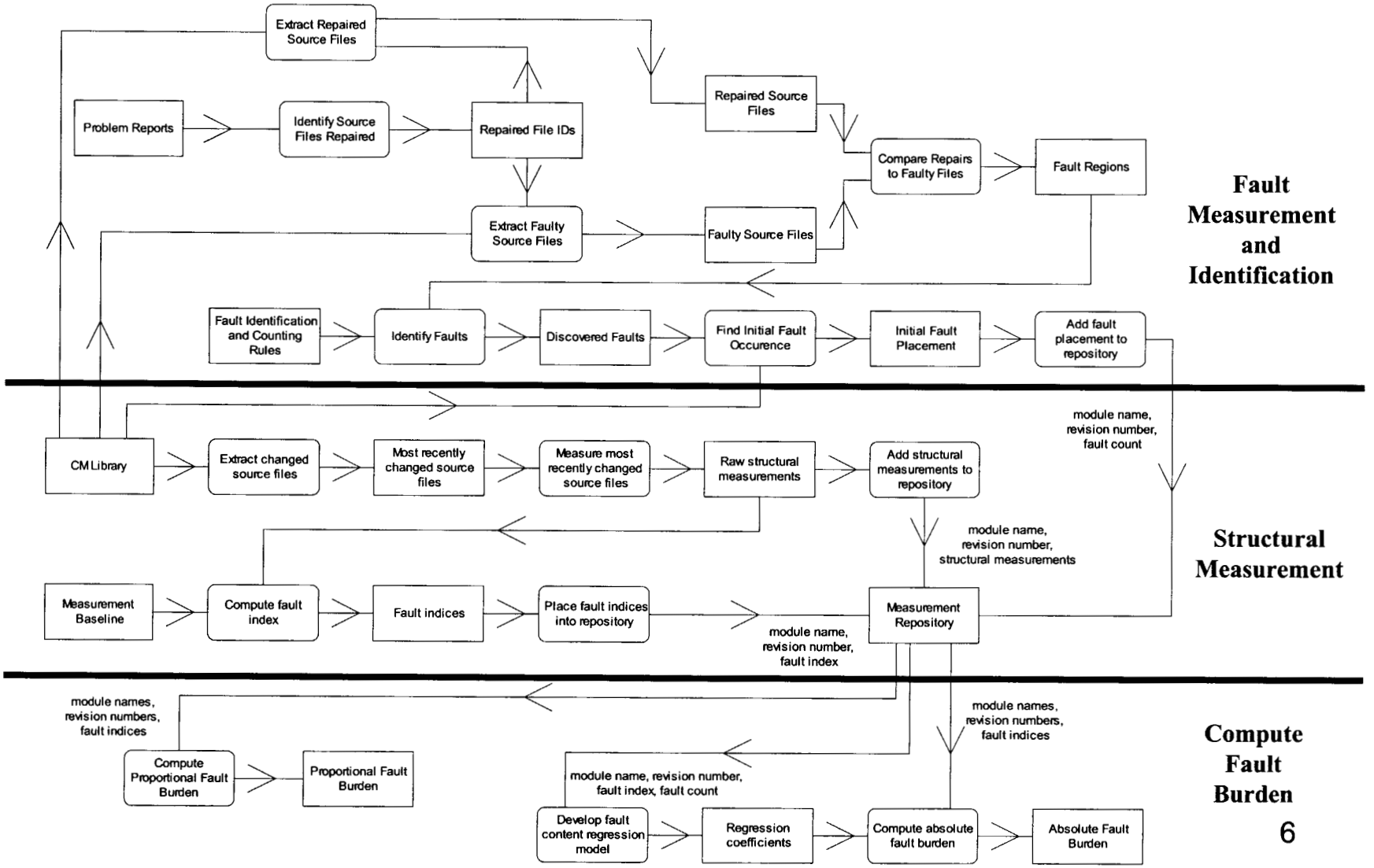
Structural Evolution at the Module Level





Motivation (cont'd)

Measurement Framework



Current State of Affairs

- No existing definition of fault in measurable terms
  - IEEE Standards
    - IEEE Std 729-1983, "IEEE Standard Glossary of Software Engineering Terminology"
    - IEEE Std 982.1-1988, "IEEE Standard Dictionary of Measures to Produce Reliable Software"
    - IEEE Std 1044-1993, "IEEE Standard Classification for Software Anomalies"
  - ODC
  - Previous work (Annual Oregon Workshop on Software Metrics, May 11-13, 1997)
  - Frankl, Hamlet, Littlewood, Stringini (IEEE TSE, vol. 24, no. 8, August 1998)

Approach

- Examine changes made in response to reported failures
- Base recognition/enumeration of software faults on the grammar of the software system's language
  - Faults found in executable, non-executable statements
- Fault measurement granularity in terms of tokens that have changed

Examples

- Example 1
  - Original statement: a = b + c * d;
  - Intended statement: a = b + c / d;
  - One token changed – "*" $\Rightarrow$ "/"
    - Coding error
  - Count number of faults as 1

Examples (cont'd)

- Example 2
  - Original statement: a = b + c * d;
  - Intended statement: a = b + (c * x) + sin(z);
  - Substantial difference between first and second statements
    - Reflects design rather than coding problem
  - Fault measurement method should reflect the degree of change

Identifying and Counting Faults

- Consider each line of text in each version of the program as a bag of tokens
  - If a change spans multiple lines of code, all lines for the change are included in the same bag
- Number of faults based on bag differences between
  - Version of program exhibiting failures
  - Version of program modified in response to failures
- Use version control system to distinguish between
  - Changes due to repair and
  - Changes due to functionality enhancements and other non-repair changes

Fault Identification Examples

- Example 1
  - Original statement: a = b + c;
    - B_1 = {<a>, <=>, <b>, <+>, <c>}
  - Modified statement: a = b - c;
    - B_2 = {<a>, <=>, <b>, <->, <c>}
  - $B_1 - B_2$ = {<+>, <-> }
  - $|B_1| = |B_2|$, $|B_1 - B_2| = 2$
  - One token has changed $\Rightarrow$ 1 fault

Fault Identification Examples (cont'd)

- Example 2
  - Original statement: a = b - c;
    - B_2 = {<a>, <=>, <b>, <->, <c>}
  - Modified statement: a = c - b;
    - B_3 = {<a>, <=>, <c>, <->, <b>}
  - $B_2 - B_3 = \{\}$
  - $|B_2| = |B_3|$, $|B_2 - B_3| = 0$
  - 1 fault representing incorrect sequencing

Fault Identification Examples (cont'd)

- Example 3
  - Original statement: a = b - c;
    - B_3 = {<a>, <=>, <c>, <->, <b>}
  - Modified statement: a = 1 + c - b;
    - B_4 = {<a>, <=>, <1>, <+>, <c>, <->, <b>}
  - $B_3 - B_4$ = {<1>, <+>}
  - $|B_3| = 6$, $|B_4| = 8$, $|B_4| - |B_3| = 2$
  - 2 new tokens representing 2 faults

Current Work

- Current Work
  - Application to JPL software development effort
    - Research
    - Production
  - Develop better models relating
    - Structural measurements of software evolution during development
    - Number and types of faults inserted